Hochiminh City University of Technology

Computer Science and Engineering

[CO1027] - Fundamentals of C++ Programming

Class

Lecturer: Duc Dung Nguyen
Credits: 3

Outcomes

❖ Understand the concept of Class.

❖ Understand advantages of Object Oriented Programming (OOP).

❖ Be able to program using OOP technique.

Outline

- ❖ Class:
  - ❖ Concept and definition
  - ❖ Encapsulation
- ❖ Constructor/Destructor

Structure versus Object-Oriented Programming

❖ **Structure programming** focuses on the process/actions that occur in a program. The program starts at the beginning, does something, and ends.

❖ **Object-Oriented programming** is based on the data and the functions that operate on it. Objects are instances of abstract data types that represent the data and its functions

Limitations of Structure Programming

❖ If the data structures change, many functions must also be changed

❖ Programs that are based on complex function hierarchies are:

  ❖ difficult to understand and maintain

  ❖ difficult to modify and extend

  ❖ easy to break

Class

❖ Class: a user defined datatype which groups together related pieces of information

  ❖ Data

  ❖ Functions (Methods)

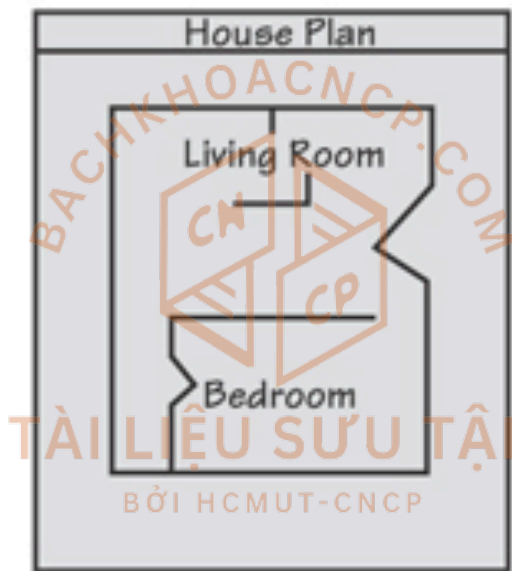❖ **Classes** are similar to **Structure** but contain functions, as well.

Terminologies

❖ Object is an instant of a particular class

❖ Data are known as fields, members, attributes, or properties

❖ Functions are known as methods

Classes and Objects

❖ A Class is like a blueprint and objects are like houses built from the blueprint

Blueprint that describes a house.

Instances of the house described by the blueprint.

Features

❖ **Encapsulation (hiding data)**: allows the programmer to group data and the subroutines that operate on them together in one place, and to hide irrelevant details from the user.

❖ **Inheritance**: allows code to be reused between related types.

❖ **Polymorphism**: allows a value to be one of several types, and determining at runtime which functions to call on based on its type.

Encapsulation

❖ Packaging related stuff together

❖ User need to know only public methods / data of the object: interface

❖ Interfaces abstract away the details of how all the operations are performed

    ❖ "Data hiding", "black box".

Class Declaration

```
class <Class_Name>
{
<access_specifier>:
    member declaration;
    ...
<access_specifier>:
    member declaration;
    ...
};
```

Class Example

```cpp
class Rectangle
{
private:
    double width;
    double height;
public:
    void setWidth(double);
    void setHeight(double);
    double getWidth();
    double getHeight();
    double getArea();
};
```

Class Access specifier

❖ Used to control access to members of the class:

  ❖ private (**default**) : the members declared as private are only accessible from within the class. No outside Access is allowed.

  ❖ public: the members declared as public are accessible from outside the Class through an object of the class.

❖ Can be listed in any order in a class

❖ Can appear multiple times in a class

Member Function Definition

❖ When defining a member function:

 ❖ Put prototype in class declaration

 ❖ Define function using class name and scope resolution operator (::)

```
void Rectangle::setWidth(double w)
{
    width = w;
}
```

Declaration vs Definition

❖ Separate the declaration (specification) part from the definition (implementation) part.

❖ Place class declaration in a header file. E.g. Rectangle.h

❖ Place member function definitions in *.cpp file. E.g. Rectangle.cpp.   This file must #include the class specification file.

❖ Programs that use the class must #include the class specification file.

Set and Get

❖ Set (mutator): a member function that stores a value in a private member variable, or changes its value in some way.

```
void setWidth(double);
void setHeight(double);
```

❖ Get (accessor): a member function that retrieves a value from a private member variable.

```
double getWidth();
double getHeight();
```

Using `const` With Member Functions

❖ `const` appearing after the parentheses in a member function declaration specifies that the function will not change any data in the calling object.

❖ Example

```cpp
double getWidth() const;

double getHeight() const;

double getArea() const;
```

Scope operator

❖ Scope operator **::**

   ❖ Is used in the definition of member function outside the class

   ❖ Inline function vs. normal function

      ❖ Member functions defined in the class definition is considered as inline function.

Static Class Members

❖ Static data members: are considered as "class" variables since they are common variables for all objects of the same class.

  ❖ Need to be initialized somewhere outside the class

  ❖ Can be accessed through object or class

  ❖ Example: object counter

❖ Static function members: can only access static members of the class.

Constructor vs Destructor

Constructor

❖ Constructors: a special function that is automatically called whenever a new object is created .

 ❖ allow the class to initialize member variables or allocate storage.

 ❖ do not return a value, including void.

 ❖ can not be called explicitly as member functions.

Default Constructor

❖ A default constructor is a constructor that takes no arguments.

❖ If you write a class with no constructor at all, C++ will write a default constructor for you, one that does nothing.

❖ A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle r;
```

Constructor Syntax

```
class <Class_Name>
{
    ...
public:
    <Class_Name>();
    ...
};
```

Constructors with Parameters

❖ To create a constructor that takes arguments:

   ❖ Indicate parameters in prototype:

```
Rectangle(double , double );
```

   ❖ Use parameters in the definition:

```
Rectangle::Rectangle(double w, double h)
{
    width = w;
    height = h;
}
```

❖ You can pass arguments to the constructor when you create an object:

```
Rectangle r2(6, 4);
```

More About Default Constructors

❖ If all of a constructor's parameters have default arguments, then it is a default constructor. For example:

```
Rectangle(double = 0, double = 0);
```

❖ Creating an object and passing no arguments will cause this constructor to execute:

```
Rectangle r;
```

Overloading Constructors

❖ A class can have more than one constructor. They can be overloaded.

❖ The compiler automatically call the one whose parameters match the arguments.

```
Rectangle();

Rectangle(double);

Rectangle(double, double);
```

Destructor

- Destructor: responsible for the necessary cleanup of a class when lifetime of an object ends.

- Destructors cannot:

  - return a value

  - accept parameters

- Destructors must have the same name as the class.

- Only one destructor per class, i.e., it cannot be overloaded

- If constructor allocates dynamic memory, destructor should release it

Destructor Syntax

```
class <Class_Name>
{
    ...
public:
    ~<Class_Name>();
    ...
};
```

Using Private Member Functions

❖ A `private` member function can only be called by another member function

❖ It is used for internal processing by the class, not for use outside of the class

❖ If you wrote a class that had a public sort function and needed a function to swap two elements, you'd make that private

Arrays of Objects

❖ Objects can be the elements of an array:

```
Rectangle rooms[8];
```

❖ Default constructor for object is used when array is defined

Arrays of Objects

❖ Must use initializer list to invoke constructor that takes arguments:

```
Rectangle rectArray[3]={Rectangle(2.1,3.2),
                        Rectangle(4.1, 9.9),
                        Rectangle(11.2, 31.4)};
```

Accessing Objects in an Array

❖ Objects in an array are referenced using subscripts

❖ Member functions are referenced using dot notation:

```
rectArray[1].setWidth(11.3);

cout << rectrArray[1].getArea();
```

Pointer to Class

❖ Objects can also be pointed by pointers. Class is a valid type.

❖ Class pointers is similar to struct pointers.

❖ E.g.:

```
Rectangle r2(6, 4);
Rectangle* r3 = &r2;
cout << r3->getArea() << endl;
cout << (*r3).getArea() << endl;
```

Using the this Pointer

❖ Every object has access to its own address through a pointer called `this` (a C++ keyword)

```
void Rectangle::setWidth(double width)
{
   this->width = width;
}
```

Summarise

❖ Understand Class: concept and definition, encapsulation

❖ Member functions, static and const members

❖ Constructor/Destructor and overloaded operators